Quinbay Technologies

Assignment – SQL and NoSQL

Author: Gayathri R

Creation Date: 24-June-2025

Contents

Introduction

This assignment explores how to work with both **SQL** and **NoSQL** databases .The goal was to get comfortable performing operations like inserting, updating, deleting, and retrieving data using two different types of databases: **MongoDB** for NoSQL and **PostgreSQL** for SQL.

By using a student management system as the base, I got to apply a variety of queries on structured and unstructured data — from simple lookups and joins to aggregations and embedded documents. Writing and testing each query helped me better understand not just the syntax, but also how each database organizes and handles data differently.

Glossary

- **SQL** – Language for working with structured data.
- **NoSQL** – Handles unstructured or flexible data.
- **PostgreSQL** – A relational database using SQL.
- **MongoDB** – A NoSQL database using documents.
- **CRUD** – Create, Read, Update, Delete operations.
- **Document** – A record in MongoDB.
- **Collection** – A group of documents in MongoDB.
- **Table** – A set of rows and columns in SQL.
- **Primary Key** – Unique ID for a row in a table.
- **Foreign Key** – Links data between tables.
- **Join** – Combines rows from multiple tables.
- **Aggregation** – Summarizes or groups data.
- **Query** – A command to get or change data.

MongoDB : mongosh version 2.5.2

Create new database: **use student_db**

1.Insert a student document.

db.students.insertOne({name: "e",roll_no: "005",email: "e@mail.com"});

```
student_db> db.students.insertOne({name: "e",roll_no: "005",email: "e@mail.com"})
{
  acknowledged: true,
  insertedId: ObjectId('68515dc35d056f3e7a50eb70')
}
student_db>
```

2.Insert an address as an embedded document inside student.

db.students.updateOne({name: "e"},{ $set: { address: { area: "abc nagar", city : "abc", state: "X" } }}

```
student_db> db.students.updateOne({ name: "e" }, { $set: { address: { area: "abc nagar", city: "abc", state: "X" } } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

3.Find a student by name.

db.students.find({ name: "a"});

```
student_db> db.students.find({ name: "a" });
[
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6c'),
    name: 'a',
    roll_no: '001',
    email: 'a@mail.com',
    phone: '1111111111',
    address: { area: 'abc nagar', city: 'abc', state: 'X' },
    enrolledCourses: [],
    marks: []
  }
]
student_db>
```

4.Update a student's phone number.

db.students.updateOne({ roll_no: "001" }, { $set: { phone: "9999999999" } });

```
student_db> db.students.updateOne(
...   { name: "a" },
...   { $set: { phone: "9999999999" } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
student_db>
```

5.Add a new course.

db.courses.insertOne({ name: "Mathematics" });

```
student_db> db.courses.insertOne({ name: "English", description: "Literature & Grammar" });
{
  acknowledged: true,
  insertedId: ObjectId('68515ec05d056f3e7a50eb71')
}
student_db>
```

6.Enroll a student in a course (push to embedded array).

db.students.updateOne({ roll_no: "001" }, { $push: { enrolledCourses: { courseId: course._id, courseName: course.name } } });

```
student_db> db.students.updateOne(
...   { name: "a" },
...   { $push: { enrolledCourses: { courseId: maths._id, courseName: maths.name } } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
student_db>
```

7.Get all students enrolled in a specific course.

db.students.find({ "enrolledCourses.courseName": "Maths" });

```
student_db> db.students.find({ "enrolledCourses.courseName": "Maths" });
[
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6c'),
    name: 'a',
    roll_no: '001',
    email: 'a@mail.com',
    phone: '9999999999',
    address: { area: 'abc nagar', city: 'abc', state: 'X' },
    enrolledCourses: [
      {
        courseId: ObjectId('68515ce05d056f3e7a50eb68'),
        courseName: 'Maths'
      }
    ],
    marks: []
  }
```

8.Add marks for a student in a course.

db.students.updateOne({ name: "a" }, { $push: { marks: courseName: “Maths”, score: 95 } } });

```
student_db> db.students.updateOne(
...   { name: "a" },
...   { $push: { marks: { courseName: "Maths", score: 95 } } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
student_db>
```

9.Get a student’s address.

db.students.findOne({ name: "a" }, { address: 1, _id: 0 });

```
student_db> db.students.findOne({ name: "a" }, { address: 1, _id: 0 });
{ address: { area: 'abc nagar', city: 'abc', state: 'X' } }
student_db>
```

10.Get all courses a student is enrolled in.

db.students.findOne({ name: "a", {enrolledCOurse: ;1, _id:0 });

```
student_db> db.students.findOne({ name: "a" }, { enrolledCourses: 1, _id: 0 });
{
  enrolledCourses: [
    {
      courseId: ObjectId('68515ce05d056f3e7a50eb68'),
      courseName: 'Maths'
    }
  ]
}
student_db>
```

11.Get marks for a specific student in a specific course.

db.students.findOne({name: “a”,”marks.courseName”: “Maths”},{“marks.$” 1});

```
student_db> db.students.findOne(
...   { name: "a", "marks.courseName": "Maths" },
...   { "marks.$": 1 }
... );
{
  _id: ObjectId('68515cfa5d056f3e7a50eb6c'),
  marks: [ { courseName: 'Maths', score: 95 } ]
}
student_db>
```

12.List all students with more than 3 enrollments.

db.students.find({ $expr: { $gt: [{ $size: { $ifNull: ["$enrolledCourses", []] } }, 3] } }, { _id: 0, name: 1, roll_no: 1 });

```
student_db> db.students.find(
...   {
...     $expr: { $gt: [{ $size: { $ifNull: ["$enrolledCourses", []] } }, 3] }
...   },
...   {
...     _id: 0,
...     name: 1,
...     roll_no: 1
...   }
... );
[ { name: 'a', roll_no: '001' }, { name: 'e', roll_no: '005' } ]
student_db>
```

13.Delete a student by ID.

db.students.deleteOne({ _id: ObjectId("68515dc35d056f3e7a50eb70") });

```
student_db> db.students.deleteOne({ _id: ObjectId("68515dc35d056f3e7a50eb70") });
{ acknowledged: true, deletedCount: 1 }
student_db>
```

14.Remove a course from student enrollment.

db.students.updateOne({ name: "b" }, { $pull: { enrolledCourses: "Biology" } });

15.Update student address city.

db.students.updateOne({ name: "a" }, { $set: { "address.city": "Chennai" } });

```
student_db> db.students.updateOne(
...   { name: "a" },
...   { $set: { "address.city": "Chennai" } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0
}
student_db>
```

16.Count number of students per course.

db.students.aggregate([{ $unwind: "$enrolledCourses" }, { $group: { _id: "$enrolledCourses", count: { $sum: 1 } } }]);

```
student_db> db.students.aggregate([
...   { $unwind: "$enrolledCourses" },
...   { $group: { _id: "$enrolledCourses", count: { $sum: 1 } } }
... ]);
[
  {
    _id: {
      courseId: ObjectId('68515ce05d056f3e7a50eb68'),
      courseName: 'Maths'
    },
    count: 1
  },
  { _id: 'Chemistry', count: 2 },
  { _id: 'English Literature', count: 4 },
  { _id: 'Maths', count: 2 },
  { _id: 'Physics', count: 2 }
]
```

17.List all students from a specific city.

db.students.find({ "address.city": "Chennai" }, { _id: 0, name: 1, roll_no: 1 });

```
student_db> db.students.find(
...   { "address.city": "Chennai" },
...   { _id: 0, name: 1, roll_no: 1 }
... );
[ { name: 'a', roll_no: '001' } ]
student_db>
```

18.Sort students by name.

db.students.find({},{ _id:0, name: 1, roll_no: 1}).sort({name:1});

```
student_db> db.students.find({}, { _id: 0, name: 1, roll_no: 1 }).sort({ name: 1 });
[
  { name: 'a', roll_no: '001' },
  { name: 'b', roll_no: '002' },
  { name: 'c', roll_no: '003' },
  { name: 'd', roll_no: '004' }
]
student_db>
```

19.Add a field to mark students as graduated.

db.students.updateMany({},{ $set: {graduated: false} });

```
student_db> db.students.updateMany({}, { $set: { graduated: false } });
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
student_db>
```

20.Get students who scored more than 90 in any course.

db.students.find({ "marks.score": { $gt: 90 } }, { name: 1, roll_no: 1, marks: 1, _id: 0 });

```
student_db> db.students.find(
...   { "marks.score": { $gt: 90 } },
...   { name: 1, roll_no: 1, marks: 1, _id: 0 }
... );
[
  {
    name: 'a',
    roll_no: '001',
    marks: [
      { course: 'Maths', score: 95 },
      { course: 'Physics', score: 91 },
      { course: 'English Literature', score: 78 }
    ]
  },
  {
    name: 'c',
    roll_no: '003',
    marks: [
      { course: 'Physics', score: 93 },
      { course: 'Maths', score: 58 }
    ]
  },
  {
    name: 'd',
    roll_no: '004',
    marks: [
      { course: 'Maths', score: 92 },
      { course: 'English Literature', score: 67 }
student_db>
  }
]
student_db>
```

21.Calculate average marks for a student.

db.students.aggregate([{ $match: { name: "a" } }, { $unwind: "$marks" }, { $group: { _id: "$name", avgMarks: { $avg: "$marks.score" } } }]);

```
student_db> db.students.aggregate([
...   { $match: { name: "a" } },
...   { $unwind: "$marks" },
...   {
...     $group: {
...       _id: "$name",
...       avgMarks: { $avg: "$marks.score" }
...     }
...   }
... ]);
[ { _id: 'a', avgMarks: 88 } ]
student_db>
```

22.Group students by city.

db.students.aggregate([{ $group: { _id: "$address.city", students: { $push: "$name" }, count: { $sum: 1 } } }]);

```
student_db> db.students.aggregate([
...   {
...     $group: {
...       _id: "$address.city",
...       students: { $push: "$name" },
...       count: { $sum: 1 }
...     }
...   }
... ]);
[
  { _id: 'Mumbai', students: [ 'c' ], count: 1 },
  { _id: 'Bangalore', students: [ 'b' ], count: 1 },
  { _id: 'Chennai', students: [ 'a' ], count: 1 },
  { _id: 'Delhi', students: [ 'd' ], count: 1 }
]
student_db>
```

23.Check if a student is enrolled in a specific course.

This query checks whether student “b” is in enrolled in “Maths” for which the output is displayed:

db.students.find({ name: "b", enrolledCourses: "Maths" }, { name: 1, enrolledCourses: 1, _id: 0 });

The query below, checks if Student “b” is enrolled in Chemistry. No output is displayed, which means that the student is not enrolled in that particular subject.

db.students.find({ name: "b", enrolledCourses: "Chemistry" }, { name: 1, enrolledCourses: 1, _id: 0 });

```
student_db> db.students.find(
...   { name: "b", enrolledCourses: "Maths" },
...   { name: 1, enrolledCourses: 1, _id: 0 }
... );
[ { name: 'b', enrolledCourses: [ 'Maths', 'English Literature' ] } ]
student_db> db.students.find( { name: "b", enrolledCourses: "Chemistry" }, { name: 1, enrolledCourses: 1, _id: 0 } );
```

24.Add multiple courses to a student.

db.students.updateOne({ name: "c" }, { $addToSet: { enrolledCourses: { $each: ["History", "Computer Science"] } } });

```
student_db> db.students.updateOne(
...   { name: "c" },
...   { $addToSet: { enrolledCourses: { $each: ["History", "Computer Science"] } } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
student_db>
```

25.Find students not enrolled in any course.

db.students.find({ enrolledCourses: { $exists: true, $eq: [] } }, { name: 1, roll_no: 1, _id: 0 });

```
student_db> db.students.find( { enrolledCourses: { $exists: true, $eq: [] } }, { name: 1, roll_no: 1, _id: 0 } );
[ { name: 'e', roll_no: '005' } ]
student_db>
```

26.List top 3 scorers in a course.

db.students.aggregate([{ $unwind: "$marks" }, { $match: { "marks.course": "Maths" } },{ $project: { name: 1, roll_no: 1, score: "$marks.score" } },{ $sort: { score: -1 } }, { $limit: 3}]);

```
[
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6c'),
    name: 'a',
    roll_no: '001',
    score: 95
  },
  {
    _id: ObjectId('68515d1b5d056f3e7a50eb6f'),
    name: 'd',
    roll_no: '004',
    score: 92
  },
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6d'),
    name: 'b',
    roll_no: '002',
    score: 76
  }
]
student_db>
```

27.Get all students and their total marks.

db.students.aggregate([{ $unwind: "$marks" }, { $group: { _id: "$name", totalMarks: { $sum: "$marks.score" }} }]);

```
student_db> db.students.aggregate([
...   { $unwind: "$marks" },
...   {
...     $group: {
...       _id: "$name",
...       totalMarks: { $sum: "$marks.score" }
...     }
...   }
... ]);
[
  { _id: 'c', totalMarks: 151 },
  { _id: 'd', totalMarks: 159 },
  { _id: 'a', totalMarks: 264 },
  { _id: 'b', totalMarks: 164 }
]
student_db>
```

28.List students along with number of courses they are enrolled in.

db.students.aggregate([{ $project: { name: 1, roll_no: 1, numCourses: { $size: "$enrolledCourses" }}}]);

```
[
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6c'),
    name: 'a',
    roll_no: '001',
    numCourses: 5
  },
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6d'),
    name: 'b',
    roll_no: '002',
    numCourses: 2
  },
  {
    _id: ObjectId('68515cfa5d056f3e7a50eb6e'),
    name: 'c',
    roll_no: '003',
    numCourses: 4
  },
  {
    _id: ObjectId('68515d1b5d056f3e7a50eb6f'),
    name: 'd',
    roll_no: '004',
    numCourses: 2
  },
  {
    _id: ObjectId('68516c135d056f3e7a50eb72'),
    name: 'e',
    roll_no: '005',
student_db>

]
```

29.Find students who failed (marks < 40).

db.students.find({ "marks.score": { $lt: 40 } }, { name: 1, roll_no: 1, marks: 1, _id: 0 });

```
student_db> db.students.find( { "marks.score": { $lt: 40 } }, { name: 1, roll_no: 1, marks: 1, _id: 0 } );
[
  {
    name: 'e',
    roll_no: '005',
    marks: [
      { course: 'Physics', score: 48 },
      { course: 'English Literature', score: 39 }
    ]
  }
]
student_db>
```

30.Delete marks for a specific course.

db.students.updateMany({ },{ $pull: { marks: { course: "English Literature" } } });

```
student_db> db.students.updateMany(
...   {},
...   { $pull: { marks: { course: "English Literature" } } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 0,
  upsertedCount: 0
}
student_db>
```

PostgreSQL : psql version 17.5

Create the necessary tables:

CREATE TABLE student (id INT PRIMARY KEY, name VARCHAR(50), email VARCHAR(100), phone VARCHAR(15));

CREATE TABLE address (student_id INT PRIMARY KEY REFERENCES student(id), street VARCHAR(100), city VARCHAR(50), state VARCHAR(50));

CREATE TABLE course (id INT PRIMARY KEY, name VARCHAR(50), description TEXT);

CREATE TABLE enrollment (student_id INT REFERENCES student(id), course_id INT REFERENCES course(id), PRIMARY KEY(student_id, course_id));

CREATE TABLE marks (student_id INT REFERENCES student(id), course_id INT REFERENCES course(id), score INT, PRIMARY KEY(student_id, course_id));

1.Insert a student.

INSERT INTO student (id, name, email, phone) VALUES (104, 'D', 'd@mail.com', '9998887777');

2.Insert student's address.

INSERT INTO address (student_id, street, city, state) VALUES (104, '4th Street', 'Hyderabad', 'TS');

3.Fetch student details by name.

SELECT * FROM student WHERE name = 'A';

```
student_management=# SELECT * FROM student WHERE name = 'A';
 id  | name |   email    |   phone
-----+------+------------+------------
 101 | A    | a@mail.com | 9999999999
(1 row)
```

4.Update student's phone number.

UPDATE student SET phone = '8888888888' WHERE id = 101;

```
student_management=# SELECT * FROM student WHERE name = 'A';
 id  | name |   email    |   phone
-----+------+------------+------------
 101 | A    | a@mail.com | 9999999999
(1 row)


student_management=# UPDATE student SET phone = '8888888888' WHERE id = 101;
UPDATE 1
student_management=# SELECT * FROM student WHERE id=101;
 id  | name |   email    |   phone
-----+------+------------+------------
 101 | A    | a@mail.com | 8888888888
(1 row)
```

5.Insert a course.

INSERT INTO course (id, name, description) VALUES (204, 'Chemistry', 'Organic and Inorganic Chemistry');

```
student_management=# INSERT INTO course (id, name, description) VALUES (204, 'Chemistry', 'Organic and Inorganic Chemistry');
INSERT 0 1
student_management=# select * from course;
 id  |   name    |           description
-----+-----------+-------------------------------
 201 | Maths     | Basic Math
 202 | Physics   | Fundamentals of Physics
 203 | English   | English Literature
 204 | Chemistry | Organic and Inorganic Chemistry
(4 rows)
```

6.Enroll a student in a course.

INSERT INTO enrollment (student_id, course_id) VALUES (104, 204);

7.Get all students enrolled in a specific course.

SELECT s.id, s.name FROM student s JOIN enrollment e ON s.id = e.student_id WHERE e.course_id = 201;

8.Insert marks for a student in a course.

INSERT INTO marks (student_id, course_id, score) VALUES (104, 204, 82);

9.Get student’s address by student ID.

SELECT * FROM address WHERE student_id = 101;

```
student_management=# SELECT * FROM address WHERE student_id = 10
 student_id |   street   |  city   | state
------------+------------+---------+-------
        101 | 1st Street | Chennai | TN
(1 row)
```

10.List courses a student is enrolled in.

SELECT c.name FROM course c JOIN enrollment e ON c.id = e.course_id WHERE e.student_id = 101;

SELECT c.name FROM course c JOIN enrollment e ON c.id = e.course_id WHERE e.student_id = 102;

```
student_management=# SELECT c.name FROM course c JOIN enrollment e ON c.id = e.course_id WHERE e.student_id = 101;
  name
---------
 Maths
 Physics
(2 rows)


student_management=# SELECT c.name FROM course c JOIN enrollment e ON c.id = e.course_id WHERE e.student_id = 102;
  name
---------
 Maths
 Physics
 English
(3 rows)
```

11.Get marks for a student in a course.

SELECT score FROM marks WHERE student_id = 101 AND course_id = 201;

```
student_management=# SELECT score FROM marks WHERE student_id = 101 AND course_id = 201;
 score
-------
    95
(1 row)
```

12.Find students with more than 3 course enrollments.

SELECT student_id FROM enrollment GROUP BY student_id HAVING COUNT(*) > 3;

```
student_management=# SELECT student_id FROM enrollment GROUP BY student_id HAVING COUNT(*) > 3;
 student_id
------------
(0 rows)
```

Initially there are no students enrolled in more than 3 courses.

INSERT INTO enrollment (student_id, course_id) VALUES (102, 204);

```
student_management=# SELECT student_id FROM enrollment GROUP BY student_id HAVING COUNT(*) > 3;
 student_id
------------
        102
(1 row)
```

13.Delete a student record.

DELETE FROM student WHERE id = 104;

Since `student_id` is referenced in other tables, directly deleting the student would cause referential conflicts. Therefore, all dependent entries must be deleted first, after which the statement `DELETE FROM student WHERE id = 104;` can be executed.

```
student_management=# DELETE FROM address WHERE student_id = 104;
DELETE 1
```

```
student_management=# DELETE FROM enrollment WHERE student_id = 104;
DELETE 1
student_management=#
```

```
student_management=# DELETE FROM marks WHERE student_id = 104;
DELETE 1
student_management=#
```

```
student_management=# DELETE FROM student WHERE id = 104;
DELETE 1
student_management=# select* from student;
 id  | name |   email    |   phone
-----+------+------------+------------
 102 | B    | b@mail.com | 8888888888
 103 | C    | c@mail.com | 7777777777
 105 | E    | e@mail.com | 6666666666
 101 | A    | a@mail.com | 8888888888
(4 rows)
```

14.Remove a course enrollment.

DELETE FROM enrollment WHERE student_id = 102 AND course_id = 203;

```
student_management=# DELETE FROM enrollment WHERE student_id = 102 AND course_id = 203;
DELETE 1
student_management=# select* from enrollment;
 student_id | course_id
------------+-----------
        101 |       201
        101 |       202
        102 |       201
        102 |       202
        103 |       202
        102 |       204
(6 rows)
```

15.Update city in address table.

UPDATE address SET city = 'xyz' WHERE student_id = 101;

```
student_management=# UPDATE address SET city = 'xyz' WHERE student_id = 101;
UPDATE 1
student_management=# select * from address where student_id=101;
 student_id |   street    | city | state
------------+-------------+------+-------
        101 | 1st Street | xyz  | TN
(1 row)
```

16.Count number of students per course.

SELECT course_id, COUNT(student_id) AS student_count FROM enrollment GROUP BY course_id;

```
student_management=# SELECT course_id, COUNT(student_id) AS student_count FROM enrollment GROUP BY course_id;
 course_id | student_count
-----------+---------------
       202 |             3
       204 |             1
       201 |             2
(3 rows)
```

17.Get all students from a particular city.

SELECT s.id, s.name FROM student s JOIN address a ON s.id = a.student_id WHERE a.city = 'xyz';

```
student_management=# SELECT s.id, s.name FROM student s JOIN address a ON s.id = a.student_id WHERE a.city = 'xyz';
 id  | name
-----+------
 101 | A
 206 | F
(2 rows)
```

18.Order students alphabetically.

SELECT * FROM student ORDER BY name;

```
student_management=# SELECT * FROM student ORDER BY name;
 id  | name |    email    |   phone
-----+------+-------------+------------
 101 | A    | a@mail.com | 8888888888
 102 | B    | b@mail.com | 8888888888
 103 | C    | c@mail.com | 7777777777
 105 | E    | e@mail.com | 6666666666
 206 | F    | f@mail.com | 9876543200
(5 rows)
```

19.Add a column 'graduated' and update values.

ALTER TABLE student ADD COLUMN graduated BOOLEAN DEFAULT FALSE; //adding column

UPDATE student SET graduated = TRUE WHERE id = 101; //setting student “101” to graduated “true”

UPDATE student SET graduated = TRUE WHERE id = 102; //setting student “102” to graduated “true”

```
student_management=# ALTER TABLE student ADD COLUMN graduated BOOLEAN DEFAULT FALSE;
ALTER TABLE
student_management=# UPDATE student SET graduated = TRUE WHERE id = 101;
UPDATE 1
student_management=# UPDATE student SET graduated = TRUE WHERE id = 102;
UPDATE 1
student_management=# select* from student;
 id  | name |    email    |   phone    | graduated
-----+------+-------------+------------+-----------
 103 | C    | c@mail.com | 7777777777 | f
 105 | E    | e@mail.com | 6666666666 | f
 206 | F    | f@mail.com | 9876543200 | f
 101 | A    | a@mail.com | 8888888888 | t
 102 | B    | b@mail.com | 8888888888 | t
(5 rows)
```

20.Find students who scored more than 90.

SELECT DISTINCT s.id, s.name FROM student s JOIN marks m ON s.id = m.student_id WHERE m.score > 90;

```
student_management=# SELECT DISTINCT s.id, s.name FROM student s JOIN marks m ON s.id = m.student_id WHERE m.score > 90;
 id  | name
-----+------
 101 | A
 102 | B
(2 rows)
```

21.Average marks per student.

SELECT student_id, AVG(score) AS average_score FROM marks GROUP BY student_id;

```
student_management=# SELECT student_id, AVG(score) AS average_score FROM marks GROUP BY student_id;
 student_id |    average_score
------------+--------------------
        101 | 91.0000000000000000
        103 | 35.0000000000000000
        102 | 78.3333333333333333
(3 rows)
```

22.Group students by city and count.

SELECT city, COUNT(*) AS student_count FROM address GROUP BY city**;**

```
student_management=# SELECT city, COUNT(*) AS student_count FROM address GROUP BY city;
   city    | student_count
-----------+---------------
 Mumbai    |             1
 xyz       |             2
 Bangalore |             1
(3 rows)


student_management=# select* from address;
 student_id |   street   |   city    | state
------------+------------+-----------+-------
        102 | 2nd Street | Bangalore | KA
        103 | 3rd Street | Mumbai    | MH
        101 | 1st Street | xyz       | TN
        206 | XYZ Street | xyz       | TN
(4 rows)
```

23.Check if a student is enrolled in a course.

SELECT * FROM enrollment WHERE student_id = 101 AND course_id = 201;
SELECT * FROM enrollment WHERE student_id = 102 AND course_id = 201;

```
student_management=# SELECT * FROM enrollment WHERE student_id = 101 AND course_id = 201;
 student_id | course_id
------------+-----------
        101 |       201
(1 row)


student_management=# SELECT * FROM enrollment WHERE student_id = 102 AND course_id = 201;
 student_id | course_id
------------+-----------
        102 |       201
(1 row)
```

24.Enroll a student into multiple courses (bulk insert).

INSERT INTO enrollment (student_id, course_id) VALUES (103, 201), (103, 203)(103,204);

25.List students with no course enrollments.

SELECT s.id, s.name FROM student s LEFT JOIN enrollment e ON s.id = e.student_id WHERE e.student_id IS NULL;

```
student_management=# SELECT s.id, s.name FROM student s LEFT JOIN enrollment e ON s.id = e.student_id WHERE e.student_id IS NULL;
 id  | name
-----+------
 105 | E
 206 | F
(2 rows)
```

26.Top 3 scorers in a course.

SELECT student_id, score FROM marks WHERE course_id = 201 ORDER BY score DESC LIMIT 3;

```
student_management=# SELECT student_id, score FROM marks WHERE course_id = 201 ORDER BY score DESC LIMIT 3;
 student_id | score
------------+-------
        101 |    95
        207 |    91
        208 |    89
(3 rows)
```

27.Students with their total marks.

SELECT student_id, SUM(score) AS total_score FROM marks GROUP BY student_id;

```
student_management=# SELECT student_id, SUM(score) AS total_score FROM marks GROUP BY student_id;
 student_id | total_score
------------+-------------
        101 |         182
        103 |          35
        102 |         235
(3 rows)
```

28.Students with number of enrollments.

SELECT student_id, COUNT(course_id) AS course_count FROM enrollment GROUP BY student_id;

```
student_management=# SELECT student_id, COUNT(course_id) AS course_count FROM enrollment GROUP BY student_id;
 student_id | course_count
------------+--------------
        101 |            2
        103 |            3
        102 |            3
(3 rows)
```

29.Students who scored less than 40 in any course.

SELECT DISTINCT student_id FROM marks WHERE score < 40;

```
student_management=# SELECT DISTINCT student_id FROM marks WHERE score < 40;
 student_id
------------
        103
(1 row)
```

30.Delete all marks for a course.

DELETE FROM marks WHERE course_id = 204;

```
student_management=# select* from marks;
 student_id | course_id | score
------------+-----------+-------
        101 |       201 |    95
        101 |       202 |    87
        102 |       201 |    76
        102 |       202 |    91
        103 |       202 |    35
(5 rows)
```